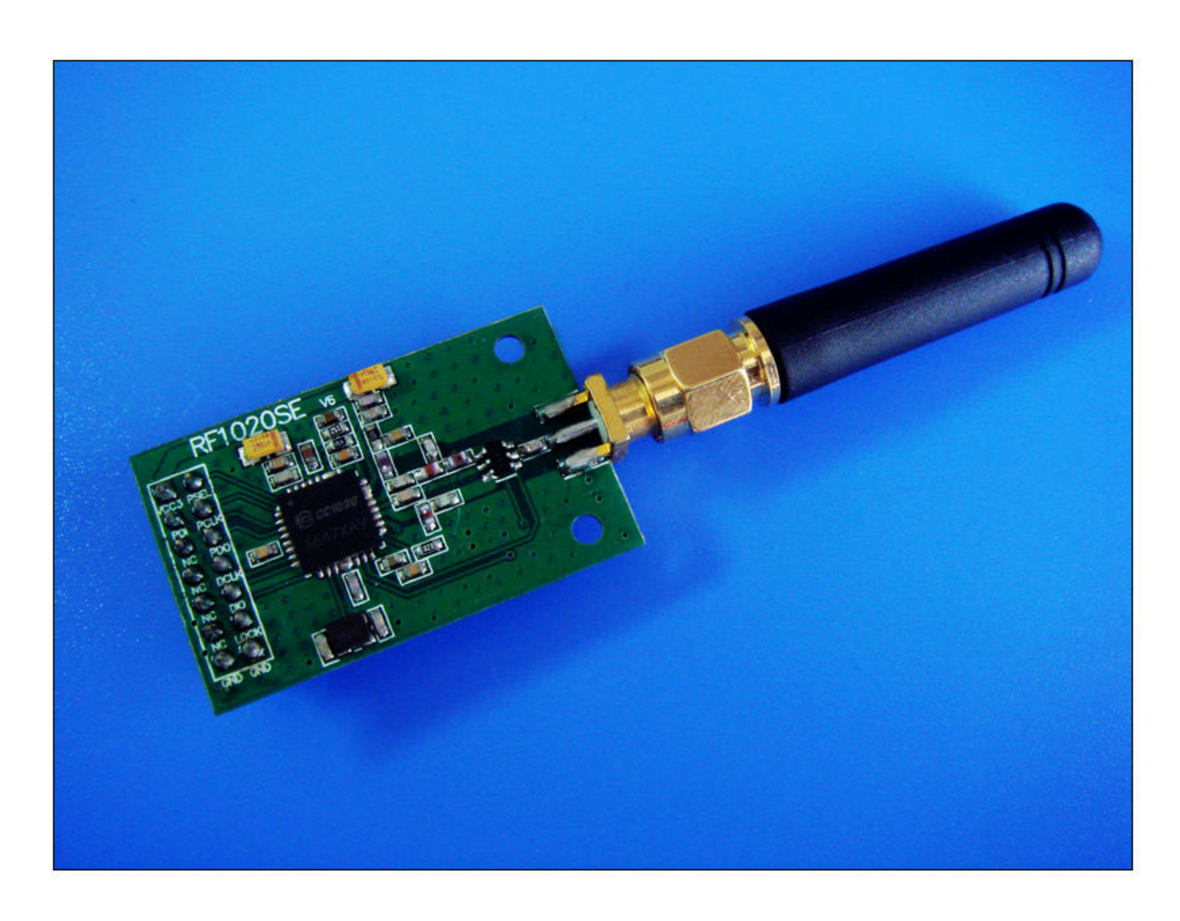
CC1020 无线模块

用户手册



CC1020SE模块(尺寸：37mm X 24mm 板厚：1mm)

联系电话：13704018223 陈 工 E-mail: chj_006@sina.com
在线咨询：QQ:35625400 474882985 MSN:1188mm88@hotmail.com

目录

产品概述

该模块使用 TI 公司的 CC1020 芯片开发而成。模块工作在 402,424, 426, 429, 433, 447,449, 469, 868 and 915 MHz 的 ISM和SRD频段,完全集成的位同步器。具有很低的IF特性，输出的解调数据在DIO引脚产生，可以通过位同步时钟DCLK读取解调数据及发送模式下的数据输出。数据包协议软件添加前导码、同步字、地址信息和发送数据长度等。可以很容易通过SPI 接口进行编程配置，电流消耗很低在发射功率为＋10dBm 时发射电流为27.1mA,接收电流为19.9mA. CC1020通信属于低速窄带传输，具有通信距离远优点。

模块特性

(1) 频率范围为402 MHz -470MHz工作

(2) 高灵敏度（对12.5kHz信道可达-118dBm）

(3) 可编程输出功率，最大10dBm

(4) 低电流消耗（RX:19.9mA）

(5) 低压供电（2.3V到3.6V）

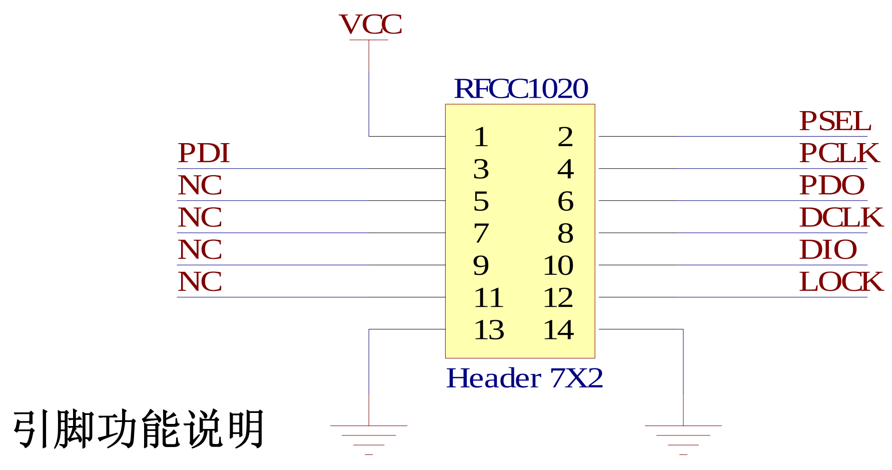
(6) 数据率最高可以达到153.6Kbaud

(7) SPI接口配置内部寄存器

(9) 标准 DIP 间距接口，便于嵌入式应用

(10)通信距离远，10dBm功率条件室外可以传输600米左右。

引脚接口说明



引脚功能说明

管脚	名称	管脚功能	备注说明
1	VCC	电源	2.3-3.6 V
2	PSEL	数字输入	SPI片选使能端，低电平有效
3	PDI	数字输入	SPI数据输入
4	PCLK	数字输入	SPI时钟
5-7-9-11	NC	无	悬空，没有使用
6	PDO	数字输出	SPI数据输出
8	DLCK	数字输出	在接收或发送模式下的同步时钟
10	DIO	输入/输出	接收模式下的输出和发送模式下的输入
12	LOCK	数字输出	PLL指示，低电平有效（可选）
13-14	GND	电源地	接地

备注

1.VCC 引脚的电压范围为2.3-3.6V 之间，不能在这个区间之外，如超过 3.6V 将会烧毁模块。推荐电压 3.3V 左右；

2.硬件没有集成SPI功能的单片机也可以控制本模块，用普通单片IO口模拟 SPI 时序进行读写操作即可；

3. 模块接口采用标准2.54mmDIP插针，13 脚、14 脚为接地脚,需要和系统电路的逻辑地连接起来；

4. 与 51 系列单片机 P0 口连接时候，需要加 10K 的上拉电阻，与其余口连接不需要。其他系列的5V单片机，如AVR、PIC，请参考该系列单片机 IO 口输出电流大小，如果超过 10mA，需要串联2-5K电阻分压，否则容易烧毁模块！如果是 3.3V 的MCU，可以直接和模块的IO口线连接。

模块电气参数

参数	数值	单位
工作电压	2.3-3.6	V
发射功率	-20-10	dBm
通信数据率	0.45-153.6	kbps
功率-20dBm时工作电流	12.3/14.5	mA
接收模式时工作电流	19.9	mA
温度范围	-40 to +85	℃
POWERDOWN 模式工作电流	1.8	uA

模块工作流程

在采用电池的应用中为了满足严格的功率消耗要求，CC1020提供很大灵活性的功率管理。POWER DOWN模式通过MAIN寄存器控制。在MAIN寄存器中有单独的位控制RX部分、TX部分、频率合成器和晶体振荡器。在每个应用中这个单独控制可用来优选最低可能电流消耗。为了阻止流入内部上拉电阻的涓流电流，在功率下降模式期间PSEL必须为高阻态，或设置为高电平。

初始化流程

当模块上电后，CC1020寄存器需要重置（通过清空MAIN寄存器的RESET位）。所有待配置的寄存器必须接着被配置（与芯片默认值不同的）。寄存器可以任何顺序自由配置。然后CC1020必须在RX和TX模式下校准。完成之后，CC1020即可被使用。

在模块上电后：

1）重置CC1020

2）初始化

3）唤起CC1020到RX

4）校准

5）唤起CC1020到TX

6）校准

在校准完成之后，进入 TX 模式（设置 CC1020TX），RX 模式（设置 CC1020RX）或 POWER DOWN 模式（设置 CC1020PD）

工作模式切换流程：

从POWER DOWN模式到RX： 1）唤起CC1020到RX 2）设置CC1020RX

从POWER DOWN模式到TX： 1）唤起CC1020到TX 2）设置CC1020TX

从RX到TX模式转换： 1）设置CC1020TX

从TX到RX模式转换： 1）设置CC1020RX

发送数据流程

当要发送数据时，首先要设置单片机中断触发方式为下降沿，连接到 RF1020 的 DIO 数据引脚为输出状态，通过 SPI 接口切换 RF1020 到发送状态。然后在 DCLK 触发的中断函数中一位一位的发送数据包。

接收数据流程

设置单片机 DIO 为输入，中断触发方式为上升沿，切换 RF1020 到接收状态，在中断函数中读取 DIO 引脚数据。当检测到数据包有效前导码、同步字后，再接收地址信息，数据包数据长度信息，接收完数据长度后接收到的就是发送的数据，把接收到的发送数据存到缓冲区，接收正确完成。接着在检测下一个数据包的前导码。

数据包协议

数据包协议为可变长度数据包，数据长度为 TX_BUFFER_SIZE，如接收到的数据大于 TX_BUFFER_SIZE，则数据包将会丢失数据。

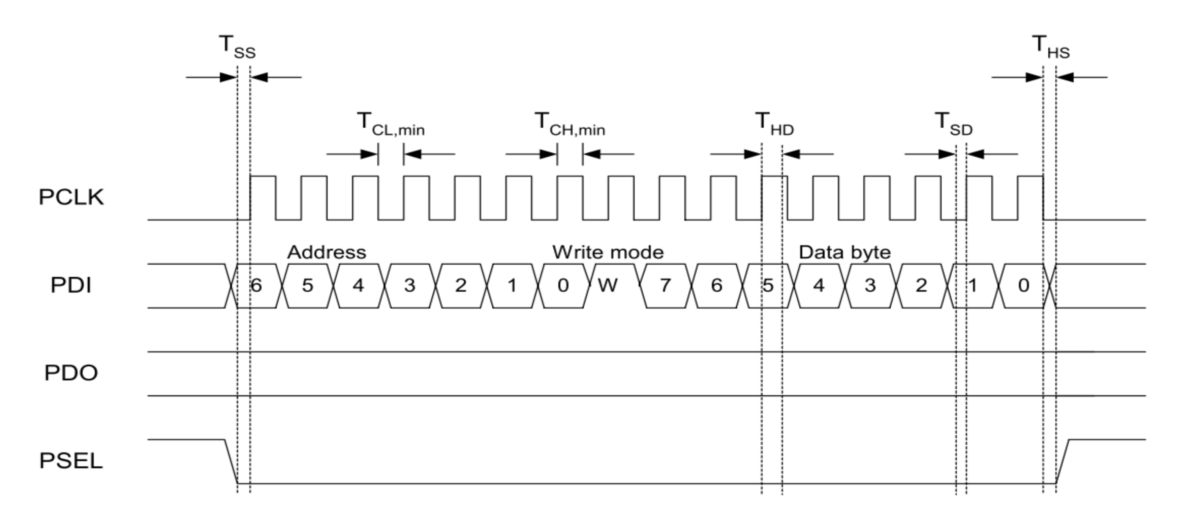
域名	字长	格式
前导码（Preamble）	4 bytes	0 1 交替出现（0x AA or 0x 55）
同步字段（SOF）	4 bytes	推荐 0x D3 0x91
地址(Address)	1 byte	1-255
数据长度（Length）	1 byte	1-255 有效数据长度
数据 Data Variable	1-255	接收到的有效数据

RF1020 模块编程指南

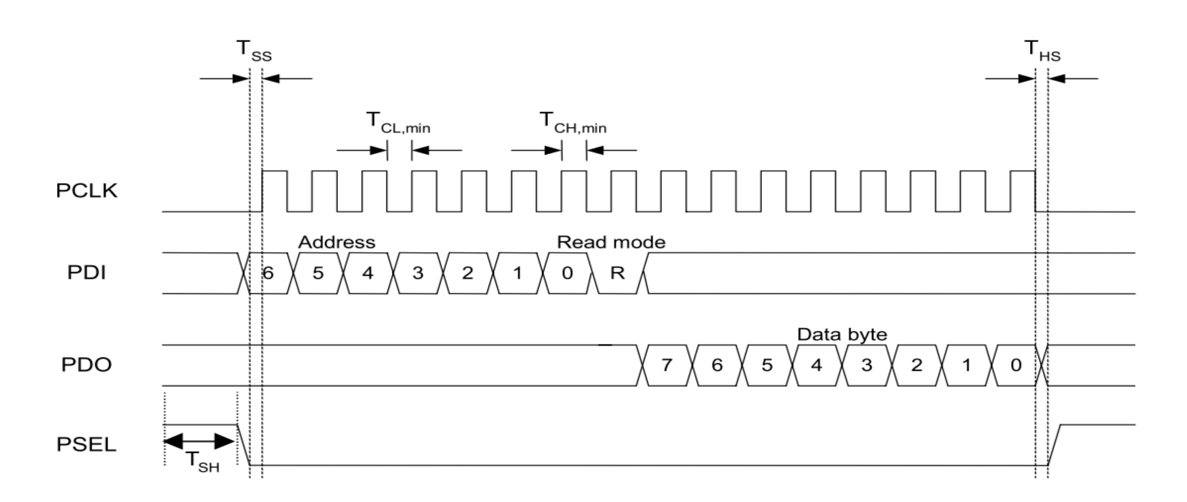
CC1020通过简单的4线SPI兼容接口（PDI,PDO,PCLK和PSEL）完成。配置寄存器为8位，每一个寄存器的读写均通过一个7位地址位、1位读/写位开始读或写操作。一个完整的配置要求的时间决定于PCLK频率。在不高于10MHz的PCLK频率下，完整的配置在小于53us内完成。在每个写周期，16个比特在PDI线上传输。每个数据帧包含7比特（A6 :0）地址位、1一个比特为R/W位（高为写，低为读）、然后8个数据比特（D7 :0）被传送。在地址和数据传送期间，PSEL必须保持为低。PDI上的数据是在PCLK 的上升沿完成所存，单片机在PCLK的下降沿建立PDI数据。在Power Down 模式下，配置寄存器的内容将保持不变。

同时，为便于用户开发，我们提供配套评估套件，为产品开发保驾护航，使无线应用开发大大加速，并避免不必要的误区。

SPI 写操作示意图



SPI 读操作示意图



工作模式寄存器介绍

CC1020 配置由设置寄存器参数来完成。配置数据的选择基于系统参数，可容易地在 SmartRF Studio 软件上得到。在 RESET 之后，所有寄存器置为默认值。TEST 寄存器不许由用户更改。为了使模块得到更好的性能，配置数据使用 SmartRF Studio 软件获得。详细寄存器介绍参考 CC1020 文档，以下为寄存器列表：

地址	名称	描述
00h	MAIN	主控制寄存器
01h	INTERFACE	接口控制寄存器
02h	RESET	数字模块复位寄存器
03h	SEQUENCING	自动加电序列控制寄存器
04h	FREQ_2A	频率寄存器 2A
05h	FREQ_1A	频率寄存器 1A
06h	FREQ_0A	频率寄存器 0A

07h	CLOCK_A	时钟产生寄存器 A
08h	FREQ_2B	频率寄存器 2B
09h	FREQ_1B	频率寄存器 1B
0Ah	FREQ_0B	频率寄存器 0B
0Bh	CLOCK_B	时钟产生寄存器 B
0Ch	VCO	VCO 电流控制寄存器
0Dh	MODEM	调制控制寄存器
0Eh	DEVIATION	TX 频偏寄存器
0Fh	AFC_CONTROL	RX AFC 寄存器
10h	FILTER	通道滤波/RSSI 控制寄存器
11h	VGA1	VGA 控制寄存器 1
12h	VGA2	VGA 控制寄存器 2
13h	VGA3	VGA 控制寄存器 3
14h	VGA4	VGA 控制寄存器 4
15h	LOCK	锁存控制寄存器
16h	FRONTEND	前端偏流控制寄存器
17h	ANALOG	模拟模块控制寄存器
18h	BUFF_SWING	LO 缓冲控制寄存器
19h	BUFF_CURRENT	LO 缓冲和偏置控制寄存器
1Ah	PLL_BW	PLL 回路带宽/负荷泵电流控制寄存器
1Bh	CALIBRATE	PLL 校准控制寄存器
1Ch	PA_POWER	功率放大寄存器
1Dh	MATCH	匹配电容阵列控制寄存器，为 RX/TX 匹配
1Eh	PHASE_COMP	LO I/Q 相位误差补偿控制寄存器

1Fh	GAIN_COMP	I/Q 混频器增益误差补偿控制寄存器
20h	POWERDOWN	掉电模式控制寄存器
21h	TEST1	PLL 校准测试寄存器
22h	TEST2	PLL 校准测试寄存器
23h	TEST3	PLL 校准测试寄存器
24h	TEST4	负荷泵和 IF 链测试寄存器
25h	TEST5	ADC 测试寄存器
26h	TEST6	VGA 测试寄存器
27h	TEST7	VGA 测试寄存器
40h	STATUS	状态寄存器（PLL 锁定、RSSI、校准就绪等）
41h	RESET_DONE	数字模块复位状态寄存器
42h	RSSI	接收信号强度指示寄存器
43h	AFC	IF 平均接收频率漂移寄存器（可 AFC 使用）
44h	GAUSS_FILTER	数字 FM 解调器寄存器
45h	STATUS1	PLL 校准结果状态（测试用）
46h	STATUS2	PLL 校准结果状态（测试用）
47h	STATUS3	PLL 校准结果状态（测试用）
48h	STATUS4	ADC 信号状态（测试用）
49h	STATUS5	通道滤波器信号‘I’状态（测试用）
4Ah	STATUS6	通道滤波器信号‘Q’状态（测试用）
4Bh	STATUS7	AGC 状态（测试用）

程序设计分析

以下介绍下驱动RF1020模块的部分核心代码，当购买我们的模块后，我们会提供完整评估完毕的测试代码，便于用户应用。同时，为便于用户开发，我们提供配套评估套件，为产品开发保驾护航，使无线应用开发大大加速，并避免不必要的误区。

SPI 写操作函数

每次对CC1020的读写操作都包含7位地址位（A6:A0）、1位写操作位（写为高）、一字节数据，写操作前必须先让PSEL使能有效（低有效）。在PCLK的上升沿，先发送地址MSB位（A6）。

```
/****************************************************
 函数功能：写一字节数据到CC1020
 入口参数：val：要写入的数据
 出口参数：无
****************************************************/
void WriteCC1020(char val)
{
 char BitCounter;
 for (BitCounter=8;BitCounter!=0;BitCounter--)
 {
    PCLK_LOW;
    PDI_LOW;
    if(val&0x80)
       PDI_HIGH;
    val<<=1;
    PCLK_HIGH;
  }
  PCLK_LOW;
}

/****************************************************
 函数功能：写CC1020寄存器值
 入口参数：addr：寄存器地址  data：写入数据
```

```
 出口参数：无
 备注    ：  无
****************************************************/
void WriteToCC1020Register(char addr, char data)
{
  PSEL_LOW;
  WriteCC1020((addr<<1)|0x01);       //写数最低位是1
  WriteCC1020(data);
  PSEL_HIGH;
}
```

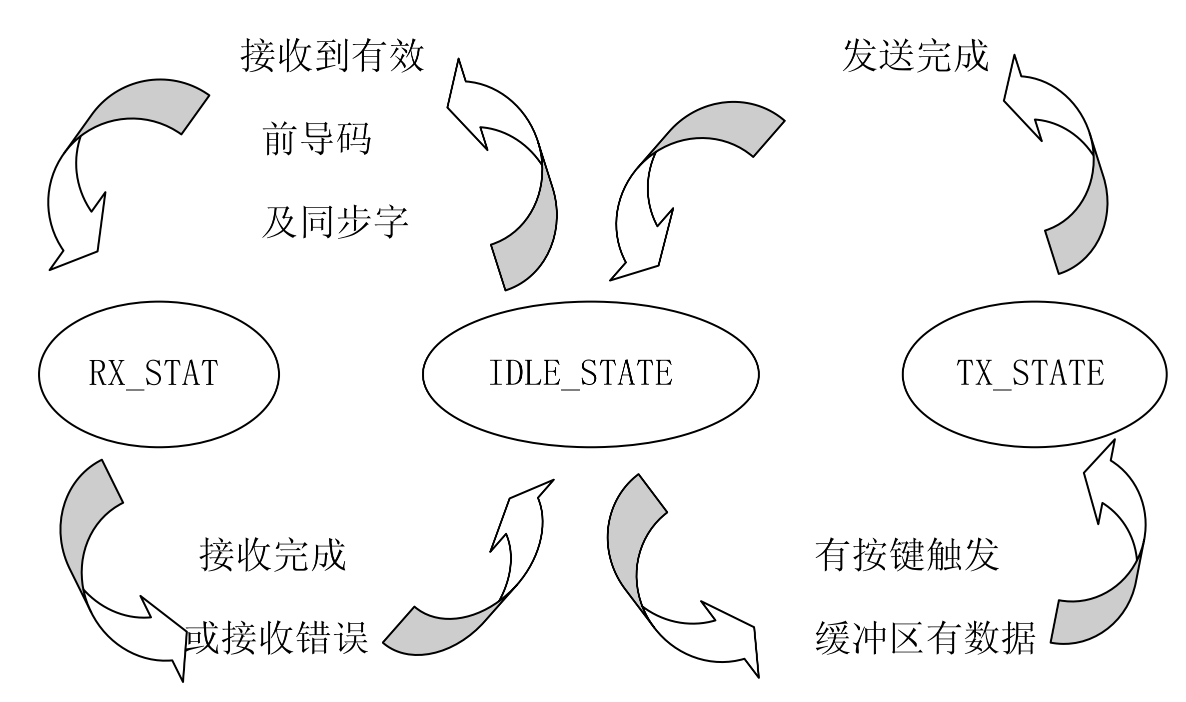
SPI 读操作函数

PCLK的上升沿，先发送读寄存器的地址MSB位，地址发送结束后发送写操作位（写操作为低），之后在PCLK的下降沿把寄存器值所存到单片机。

```
/****************************************************
 函数功能：读cc1020寄存器值
 入口参数：addr：寄存器地址
 出口参数：返回寄存器内容
 备注    ：  无
****************************************************/
char ReadFromCC1020Register(char addr)
{
  char BitCounter;
  char Byte;
  PSEL_LOW;
WriteCC1020(addr<<1);        // Send address bits
//CC1020读操作 发送7Bit地址 1Bit R/W 0-W，位移后最低位一定是0。
 for(BitCounter=8;BitCounter!=0;BitCounter--)
 {
      PCLK_HIGH;
      Byte<<=1;
      if(PDO_IN)
      Byte|=1;
      PCLK_LOW;
 }
 PSEL_HIGH;
 return Byte;
```

}

状态切换示意图



程序设计采用状态机切换方式，主程序处理发送缓冲区数据及把从NewMsg-RF1020接收到接收缓冲区的数据通过UART发送到PC机显示。当没有按键触发发送数据及接收缓冲区为空的时候状态机处于IDLE_STATE状态，模块不断检测数据包前导码，如果检测到正确的前导码及一字节同步字，状态切换到RX_STATE接收同步字、地址、数据包长度、数据。如果有按键触发，主程序初始化发送缓冲区数据包长度及装载发送数据，切换到TX_STATE发送数据包数据，先发送前导码。

状态切换函数

```
/*****************************************************
函数功能：维护状态
入口参数：无
```

出口参数：无
备注　　：空闲状态下 CC1020 为接收模式，通过中断查询前导码
```
**************************************************/
void ChangeState(void)
{
  switch(NextState)                    //下一个状态
  {
    case RX_STATE:                     //下一状态为接收
      if(State==TX_STATE)              //当前为发送
      {
          TI_CC_DCLK_PxIES &= ~DCLK;     // Int on raising edge
          TI_CC_DIO_PxDIR     &=  ~DIO;         //  Set  DIO  as  input
SetupCC1020RX(RXANALOG, PA_POWER);//发送到接收模式切换
       }
      State=RX_STATE;                   //接收状态
      BitCounter =0;                    //位计数器清零
      ByteCounter=0;                    //字计数器清零
      break;
    case TX_STATE:                  //下一个状态为发送状态
      if(State!=TX_STATE)           //当前不是发送状态
      {
        TI_CC_DCLK_PxIES |= DCLK;   //INT on falling edge
        TI_CC_DIO_PxDIR  |= DIO;   // Set DIO as output
SetupCC1020TX(TXANALOG, PA_POWER);  //从接收到发送切换
      }
      State=TX_STATE;  //状态为发送
BytesToSend=TXBufferIndex;
// Number of bytes to send  发送字节数 包含 Preable 和 Header
  TXBuffer[PreambleLength+3]=BytesToSend-HEADER_SIZE-PreambleLength;
//装载数据长度
      LastDataBit = FALSE;          //标志清零
      TXBufferIndex=0;              //发送索引清零
      BitCounter=0;                 //位计数器清零
      ShiftReg=TXBuffer[TXBufferIndex++];  //装载发送数据    装载第一
个发送数据
      printfch(ShiftReg);////////////////
      break;
    case IDLE_STATE: //下一状态为空闲
      if(State==TX_STATE)
      {      //  当前状态为发送
        TI_CC_DCLK_PxIES &= ~DCLK; // Int on raising edge
        TI_CC_DIO_PxDIR  &= ~DIO; // Set DIO as input
        SetupCC1020RX(RXANALOG, PA_POWER);      //  从发送到接收切换
      }
```

```
        State=IDLE_STATE;    //当前状态为空闲
        //空闲状态初始化发送缓冲区下标
        TXBufferIndex=HEADER_SIZE+PreambleLength;
        PreambleCount=0;
        PreambleError=0;
        PreambleFound=FALSE; // Preamble 头标志清零
        UI1Found=FALSE;    // UI1 标志清零
        break;
    }
}
```

中断方式接收或发送数据

CC1020 和单片机的通信采用中断方式，在 DLCK 产生的中断函数中通过 DIO 完成通信。首先，定义了一个字节的移位寄存器共用体：

```
union
{
        char ShiftReg;                //发送，接收字节
        struct
{
                unsigned char ShiftRegLSB :1;
                unsigned char :1;
                unsigned char :1;
                unsigned char :1;
                unsigned char :1;
                unsigned char :1;
                unsigned char :1;
                unsigned char ShiftRegMSB :1;
 };
};

/*****************************************************************
 函数功能：CC1020 收发数据中断处理
 入口参数：无
 出口参数：无
 备注   ：  空闲状态下先接收前导码
*****************************************************************/
#pragma vector=PORT1_VECTOR
__interrupt void PORT1_ISR(void)
```

```
{
  switch (State)
  {
    case TX_STATE:
      // Write data to CC1020
      //DIO=ShiftRegMSB;
      if (ShiftRegMSB)   // MCU 发送数据到 CC1020
            TI_CC_DIO_PxOUT |=   DIO;
      else
            TI_CC_DIO_PxOUT &= ~ DIO;
      ShiftReg=ShiftReg<<1;
      BitCounter++;
      // If last data bit has been sent
      if(LastDataBit) //是否是最后一个数据
      {
          // Remain in TX until last data bit has been sent on RF:
          if(BitCounter > RFPACKET_EXTENSION)
       //发送最后 RFPACKET_EXTENSION bit 位 保持输出功率
          {
            BitCounter    = 0;
            TXBufferIndex = 0;
            LastDataBit = FALSE;
            NextState=IDLE_STATE;
            break;
          }
          else
          {             }
      // Else (not last data bit)
      }
      else
      {
           // Load new TX data and monitor end of packet:
           //装载新的发送数据 并检测是不是最后一个数据
          if(BitCounter==1)
          {
            if(TXBufferIndex>BytesToSend)  //是否发完最后一个发送数据
            {
              BitCounter = 0;
              LastDataBit=TRUE;          //最后字节数据标志
              ShiftReg=0x00;  //发送完数据 发送 0x00 作为结束标志
            }
          }
             if(BitCounter==8) //发完一个字节
             {
```

```
                BitCounter=0;
                ShiftReg=TXBuffer[TXBufferIndex++];
            }
        }
    break;
        case RX_STATE:
      // Read data from CC1020
      ShiftReg=ShiftReg<<1;
      ShiftRegLSB= ((TI_CC_DIO_PxIN & DIO)>>1);   // Low-side LO (DIO
not inverted)      //移位到最后一位
      BitCounter++;
      // If received 8bits=1byte
      if(BitCounter==8)  //接收到一个字节
      {
        BitCounter=0;
       // Process received RF data:
       switch(ByteCounter)
       {
         // Byte-0 = SOF part 1:
         case 0 :
           if(ShiftReg!=UI2) //UI2 接收不正确回到 空闲状态
           {
               NextState=IDLE_STATE;
           }
           break;
           // Byte-1 = address:
         case 1 :     //接收到地址字节
           // Addressing not implemented
           break;
           // Byte-2 = packet length/size:
         case 2 : //数据包长度
           BytesToReceive=ShiftReg; //为长度减 1
           if(BytesToReceive>TX_BUFFER_SIZE)
              //接收到的数据长度大于 缓冲区长度
           {
             BytesToReceive=0;
           }
           break;
         // Rest of the packet is data, store it in the receive buffer
         default :
           RXBuffer[RXBufferWriteIndex]=ShiftReg;
          //存入缓冲区
     //RXBufferWriteIndex=(RXBufferWriteIndex+1)%RX_BUFFER_SIZE;
           RXBufferWriteIndex++;
```

```
            RXBufferWriteIndex&=0x3F;
            break;
        }
        if(ByteCounter>=BytesToReceive+2)    //如果接收数据包完成 //
        {
          NextState=IDLE_STATE; //转到空闲状态
         }
        ByteCounter++;
      }
      break;
    case IDLE_STATE: //
      // Read data from CC1020
      ShiftReg=ShiftReg<<1; //空闲状态小 从读 CC1020 数据
      //ShiftRegLSB=DIO;   // Low-side LO (DIO not inverted)
      ShiftRegLSB= ((TI_CC_DIO_PxIN & DIO)>>1);       //读 DIO P2.4
      BitCounter++;
      // If preamble found, look for Start Of Frame (SOF)
      if(PreambleFound)       //前导标志置 1  受到有效前导
      {
// If first unique identifier found, enter RX mode
  if(ShiftReg==UI1)         //UI1 被找到
  {
      // Initialise RX processing state directly:
     // Avoid latency with background scheduler.
     BitCounter=0;  //
   ByteCounter=0;
    State = RX_STATE;   //收到一个则进入接收状态
   NextState=RX_STATE;
// Else if we are still receiving preamble, do nothing
} //没有正确收到 UI1
else
if((ShiftReg==VALID_PREAMBLE_BYTE_1)||(ShiftReg==VALID_PREAMBLE_BYTE_
2)) //如果仍然在收到前导 什么都不做
{
  // Else if we are not receiving a correct preamble, declare an error
  }else if(PreambleError==0)       //否则前导错误
{
PreambleError++;     //前导错误变量加 1
  }else
  {
  }
// If preamble error found, increase the error counter regardless of bits
read
if(PreambleError>0)
```

```
{           //前导错误变量加 1
      PreambleError++;
}
// Once an error condition has occurred, a correct SOF must be found
// within 9 bits (error counter is initially incremented by 2), otherwise
// we abort and start looking for preamble again
if(PreambleError>10)     //如果大于 10
  {
PreambleFound=FALSE;  //前导错误标志 复位
  }
// Else (preamble has not been found) //前导标志没有置位
      }
else
      {   //从 CC1020 读取值    首先读取前导
         // If valid preamble, increase counter
if(ShiftReg==VALID_PREAMBLE_BYTE_1)||(ShiftReg==VALID_PREAMBLE_BYTE_2
))  //如果读取到有效的 Preamble
{
     PreambleCount++;      // 读到前导 Preamble 加 1
// Else (not valid preamble), reset counter
}else
{
    PreambleCount=0; //如果收到有一次错误 Preamble 则 前导码计数器清零
}
// If required preamble reached, indicate preamble found
if(PreambleCount>=PREAMBLE_REQ) // 指示 Preamble 找到
{
    PreambleFound=TRUE;
    //置位前导标志
   PreambleError=0;
  }
//printfch(PreambleCount);
  }
  break;
     default:
   // Enter known state in case something goes haywire
NextState=IDLE_STATE;
  break;
  }
  //INTF=0;
  TI_CC_DCLK_PxIFG &= ~DCLK;               // Clear flag
}
```

无线应用注意事项

(1) 无线模块的 VCC 电压范围为 1.8V-3.6V 之间，不能在这个区间之外，超过 3.6V 将会烧毁模块。推荐电压 3.3V 左右。

(2) 除电源 VCC 和接地端，其余脚都可以直接和普通的 51 单片机 IO 口直接相连，无需电平转换。当然对 3V 左右的单片机更加适用了。

(3) 硬件上面没有 SPI 的单片机也可以控制本模块，用普通单片机 IO 口模拟 SPI 不需要单片机真正的串口介入，只需要普通的单片机 IO 口就可以了，当然用串口也可以了。模块按照接口提示和母板的逻辑地连接起来

(4) 标准 DIP 插针，如需要其他封装接口，或其他形式的接口，可联系我们定做。

(5) 任何单片机都可实现对无线模块的数据收发控制，并可根据我们提供的程序，然后结合自己擅长的单片机型号进行移植;

(6) 频道的间隔的说明：实际要想 2 个模块同时发射不相互干扰，两者频道间隔应该至少相差 1MHZ，这在组网时必须注意,否则同频比干扰。

(7) 实际用户可能会应用其他自己熟悉的单片机做为主控芯片，所以，建议大家在移植时注意以下 4 点：

A:确保 IO 是输入输出方式，且必须设置成数字 IO;

B:注意与使用的 IO 相关的寄存器设置，尤其是带外部中断、

带 AD 功能的 IO，相关寄存器一定要设置好;

C:调试时先写配置字，然后控制数据收发

D:注意工作模式切换时间

我们的承诺

最后，欢迎您使用我们的产品，在应用中有技术问题请及时向我们联系，我们会予以技术知道，同时运输中出现产品问题我们会全面责任并予以更换。

愿与您一起走向成功